|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| logowydzialu | Instytut Informatyki Politechniki Śląskiej  Zespół Mikroinformatyki i Teorii Automatów Cyfrowych  **Projekt BIAI** | | | logoii | |
| **Rok akademicki:** | **Rodzaj studiów\*: SSI/NSI/NSM** | **Przedmiot:** | | **Grupa:** | **Sekcja:** |
| 2016/2017 | SSI | BIAI | | BDIS | 3 |
| **Skład sekcji:** | Sebastian Oprzędek | **Prowadzący**:  OA/JP/KT/GD/BSz/GB | | GB | |
|  | Krzysztof Kundera |
| *Sprawozdanie* | | | | | |
| Temat:  Szacowanie zmiany położenia punktu na podstawie dotychczasowej trajektorii ruchu (śledzenie obiektów) | | | | | |
| Data:  dd/mm/yyyy | | | 30/07/2017 | | |

1. *Temat projektu.*

Szacowanie zmiany położenia punktu na podstawie dotychczasowej trajektorii ruchu (śledzenie obiektów)

1. *Założenia projektu.*

- Wyznaczanie kolejnych punktów poruszania się obiektu na podstawie

dotychczasowej trajektorii w przestrzeni dwuwymiarowej za pośrednictwem sieci neuronowej

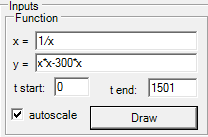
- Możliwość konfiguracji parametrów sieci (ilość warstw, ilość neuronów na

warstwach)  
- Analiza wyników dla różnych konfiguracji sieci  
- Prezentacja wyników w postaci wykresu  
- Porównanie wyników z klasyczną aproksymacją funkcjami (np. potęgowa,

wielomiana, wykładnicza)

1. *Analiza zadania.*

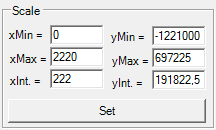
Projekt powinien być łatwy w obsłudze dla użytkownika oraz umożliwiać jak największą automatyzację zadań, tak aby wykonanie testów wymagało jak najmniejszej ilości akcji przez użytkownika. Projekt powinien posiadać gui umożliwiające wykonanie założeń wraz z rysowaniem wykresów.

1. *Specyfikacja zewnętrzna.*
2. Rysowanie funkcji

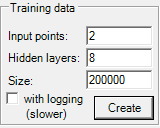
Ramka umożliwiająca rysowanie funkcji  
w postaci x(t) i y(t) na wspólnym wykresie

umieszczonym poniżej. Funkcja rysowana jest  
w zakresie od t-start do t-end. Dla każdego  
punktu z tego zakresu rysowany jest jeden  
punkt na wykresie. Dołączona jest także opcja  
autoskalowania wykresu.

Dozwolone są tutaj operacje mnożenia, dzielenia i dodawania oraz proste funkcje – sin, cos, log, tan, abs.

1. Skalowanie wykresu

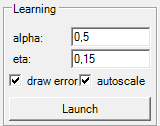
Ramka umożliwia skalowanie wykresu zgodnie z podanymi parametrami.

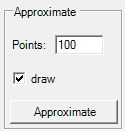
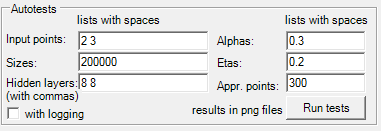


1. Generowanie danych uczących

Ramka umożliwia generowanie danych testowych  
zgodnie z podanymi parametrami.  
Input points – liczba par x,y branych pod uwagę  
podczas generowania danych  
Hidden layers – liczba neuronów na warstwach  
wewnętrznych. Warstwy oddzielane są za pomocą spacji

1. Uczenie sieci

Ramka umożliwia uczenie sieci stworzonymi danymi testowymi  
z podanymi parametrami alpha i eta.  
Opcja draw error pozwala stworzyć wykres błędu, a opcja autoscale pozwala na autoskalowanie tego wykresu.**

1. Aproksymacja (testowanie sieci)  
   Ramka pozwala na aproksymacje funkcji podanej w ramce odpowiedzialnej za rysowanie za pomocą sieci nauczonej danymi uczącymi.  
   Wynik aproksymacji jest dołączany do wykresu (jeśli oznaczony jest checkbox „draw”)
2. *Autotestowanie*

Autotestowanie umożliwia wykonanie wszystkich powyższych akcji dla podanych parametrów   
w odpowiedniej kolejności (rysowanie funkcji, generowanie danych, uczenie i aproksymacja). Otrzymany wykres zapisywany jest do pliku graficznego, a w nazwie znajdują się wszystkie parametry sieci wraz z otrzymanym pierwiastkiem błędu średniokwadratowego.   
Parametry oddzielane są za pomocą spacji, za wyjątkiem „hidden layers” gdzie spacjami są oddzielane wartstwy w pojedynczej konfiguracji, a konfiguracje oddzielane są przecinkami.

1. *Specyfikacja wewnętrzna.*
2. Sieć neuronowa  
     
   Własna implementacja sieci neuronowej podzielona jest na 3 klasy główne:  
   - Net

- Neuron

- TrainingData

1 pliki pomocnicze:

- Connection

- Topology

oraz klasę kontrolującą sieć:

- NeutralNetwork

Net – klasa główna, zawierająca model sieci (strukturę), ostatnie błędy uczenia oraz standardowe funkcje sieci neuronowych: feedForward, getResults, backProp oraz getRecentAverageError. Dołączona jest także funkcja zapisująca sieć do pliku „net.txt”

class Net

{

private:

Topology topology;

double prv\_error;

double prv\_recentAverageError;

double prv\_recentAverageSmoothingFactor;

};

public:

Net(const TopologySchema &topologySchema, double alpha, double eta);

Net(string fileName);

~Net();

void feedForward(const vector<double> &inputValues);

void backProp(const vector<double> &targetValues);

void getResults(vector<double> &resultValues) const;

double getRecentAverageError(void) const { return prv\_recentAverageError; }

void save(string filename);

}

TrainingData – klasa używana do tworzenia pliku z danymi uczącymi oraz następnie pobierania tych danych. Dane przechowywane są w pliku „trainingData.txt”.

enum OpenType{READ, WRITE};

class TrainingData

{

fstream prv\_trainingDataFile;

public:

TrainingData(const string filename, OpenType openType);

~TrainingData();

bool isEof(void) { return prv\_trainingDataFile.eof(); }

void getTopology(vector<unsigned> &topology);

void setTopology(TopologySchema topologySchema);

unsigned getNextInputs(vector<double> &inputValues);

unsigned getTargetOutputs(vector<double> &targetOutputValues);

void generate(TopologySchema topologySchema, int size, int tStart, int tEnd, int tDelta, string xFunction, string yFunction);

};

Neuron – Klasa przechowująca pojedynczy neuron. Przechowuje aktualne wartości neuronu oraz umożliwia operacje na nich. Każdy neuron połączony jest strukturą Connection z neuronami z kolejnej warstwy, aby umożliwić operacje na strukturze warstwowej.

typedef vector<Neuron> Layer;

class Neuron

{

const unsigned prv\_myIndex;

double prv\_outputValue;

vector<Connection> prv\_outputWeights;

static double transferFunction(double x);

static double transferFunctionDerivative(double x);

double prv\_gradient;

public:

Neuron(unsigned howManyOutputs, const unsigned \_myIndex, double \_alpha, double \_eta);

~Neuron();

void setOutputValue(double outValue) { prv\_outputValue = outValue; }

double getOutputValue(void) const { return prv\_outputValue; }

void feedForward(const Layer &previousLayer);

void calcOutputGradients(double targetValue);

void calcHiddenGradients(const Layer &nextLayer);

double sumDOW(const Layer &nextLayer) const;

void updateInputWeights(Layer &prevLayer);

double alpha; //0.0 - n multiplier of the last weight change (momentum)

double eta; //0.0 - 1.0 training rate

static double randomWeight(void) { return rand() / double(RAND\_MAX);}

string toString();

void update(vector<double> values);

};

Connection – struktura przechowująca wagi połączeń pomiędzy neuronami z sąsiednich warstw.

struct Connection

{

double weight;

double deltaWeight;

};

Topology – plik przechowujący funkcje pomocnicze służące głównie do ułatwienia pracy na warstwach neuronów.

typedef vector<unsigned> TopologySchema;

typedef vector<Neuron> Layer;

typedef vector<Layer> Topology;

Topology createTopology(TopologySchema topologySchema, double alpha, double eta);

Topology createTopology(TopologySchema topologySchema);

TopologySchema createTopologySchema(string topologySchemaString);

TopologySchema createTopologySchema(int inputPoints, string hiddenLayers);

TopologySchema getTopologySchemaFromFile(fstream &file);

vector<TopologySchema> createTopologySchemas(vector<unsigned> inputs, vector<string> hiddenLayers);

string toString(TopologySchema);

TopologySchema toTopologySchema(Topology topology);

NeutralNetwork – klasa kontrolująca sieć neuronową. Umożliwia operacje na klasach wymienionych powyżej:  
- tworzenie danych testowych  
- uczenie sieci  
- aproksymacja dla poszczególnego punktu oraz dla przedziału.

vector<double> getInputsForPoint(string xFunction, string yFunction, int inputPoints, string hiddenLayers, double t, Normalizer xNormalizer, Normalizer yNormalizer);

void approximate(int size, int tStart, int tEnd, stringstream& ss, string xFunction, string yFunction, TopologySchema topologySchema, int points, ChartArea^ chartArea, Series^ chartSeries, bool draw, double& rootMeanSqureErrorX, double& rootMeanSqureErrorY);

void launchLearning(double alpha, double eta, stringstream& ss, ChartArea^ chartArea, Series^ chartSeries, bool draw, bool autoscale);

void createTestData(TopologySchema topologySchema, int size, int tStart, int tEnd, int points, string xFunction, string yFunction, System::Windows::Forms::TextBox^ textBox, bool withTopology);

1. Common

Moduł zawierający funkcje, które mogą być ponownie wykorzystane w innym projekcie.  
Zawartość:  
- zewnętrzna biblioteka expression\_parser – umożliwiająca rozwiązywanie prostych równań.  
- klasa Expression  
- klasa Normalizer  
- klasa StringHelper

Klasa Expression – klasa rozszerzająca możliwości biblioteki expression\_parser o podmienianie parametru p w równaniu o podaną wartość oraz rozwiązująca równania z nawiasami oraz proste funkcje: sin, cos, tan, abs i log.

typedef string Function;

void replaceStrings(const std::string& str, const std::string& toReplace, const std::string& replacer);

Function replaceParameterT(Function expression, double parameter);

Function commonFunctions(Function expression);

Function solveBrackets(Function expression);

double solve(Function expression);

double solve(Function expression, double parameterT);

double maxValue(Function function, int tStart, int tEnd);

double minValue(Function function, int tStart, int tEnd);

Normalizer – klasa normalizująca wartości do przedziału 0-1 i spowrotem.

class Normalizer

{

double scale;

double minValue;

public:

Normalizer(double minValue, double maxValue);

Normalizer(double scale);

double normalize(double realValue);

double realValue(double normalizedValue);

};

StringHelper – plik przechowujący funkcje umożliwiające operacje na ciągach znaków.

void replaceStrings(std::string& str, const std::string& toReplace, const std::string& replacer);

vector<string> splitByChar(const string str, const char c);

vector<string> splitBySpaces(const string& str);

vector<string> splitBySpaces(System::String^ systemString);

vector<string> splitByCommas(const string str);

vector<string> splitByCommas(System::String^ systemString);

vector<unsigned> toUnsignedIntVector(string line);

vector<unsigned> toUnsignedIntVector(System::String^ systemString);

vector<double> toDoubleVector(string line);

vector<double> toDoubleVector(System::String^ systemString);

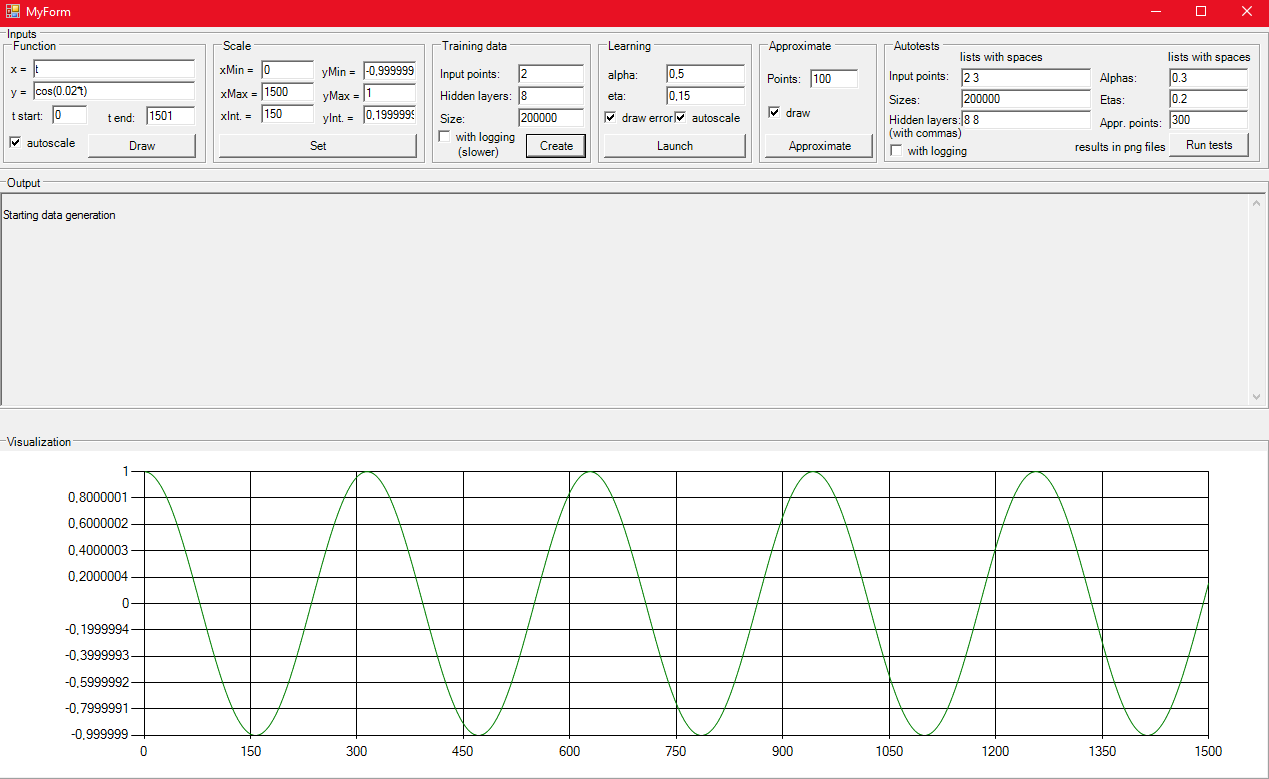
string toStdString(System::String^ systemString);

System::String^ toSystemString(string s);

string showVectorVals(string label, std::vector<double> &v);

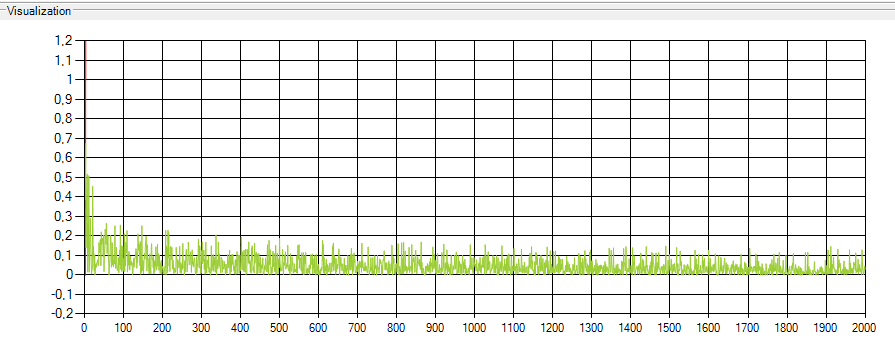
1. Klasa gui – MainForm

Udostępnia widok dla użytkownika oraz kontrolki pozwalające sterować programem.  
Z tego poziomu kontrolowana jest cała aplikacja wraz wykresem.



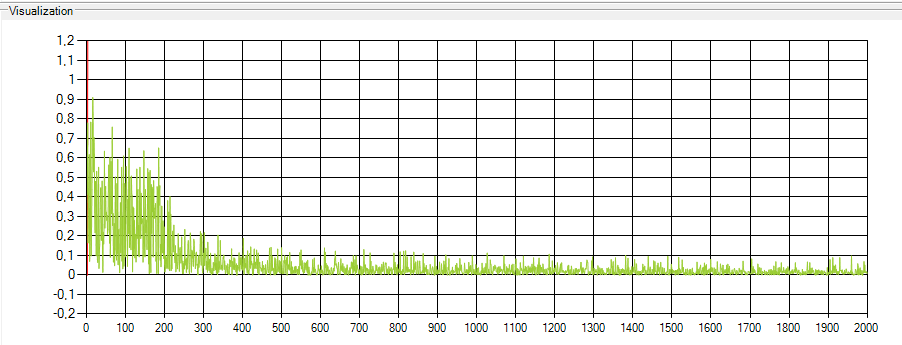
1. Wnioski z okresu tworzenia programu
2. Ilość warstw neuronów

1 . Testy przeprowadzone na funkcji liniowej y=ax+b, gdzie a i b są losowe, a przyrost x jest stały.  
dane wejściowe sieci: 5 kolejnych współrzędnych y  
dane wyjściowe sieci: 6 współrzędna y  
  
Wykres błędu dla pliku:   
trainingYCoordinate.txt - wrong, too small net   
Topologia: 5 4 1



Wynik jest niezadowalający. Końcowy błąd jest zbyt duży.

Wykres błędu dla pliku:   
trainingYCoordinate.txt – good  
Topologia: 5 4 4 1



Końcowy błąd jest mniejszy. Efekt został poprawiony przez dodanie drugiej warstwy ukrytej w sieci.

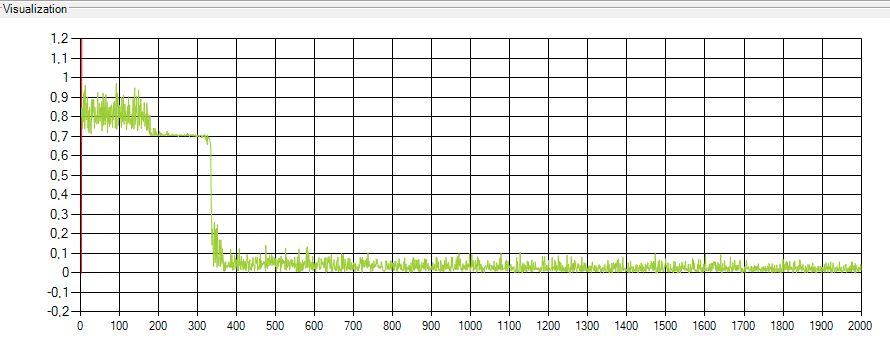
Wnioski:

* Sieć z większą liczbą warstw uczy się dłużej, ale efekt końcowy jest lepszy.
* Dla naszych potrzeb sieć jednowarstwowa prawdopodobnie nie wystarczy.

2. Testy dla funkcji liniowej w postaci:

x = a\*t  
y= b\*t  
gdzie a i b są losowe, a przyrost t jest stały   
dane wejściowe sieci: 5 kolejnych współrzędnych x i y  
dane wyjściowe sieci: 6 współrzędne x i y

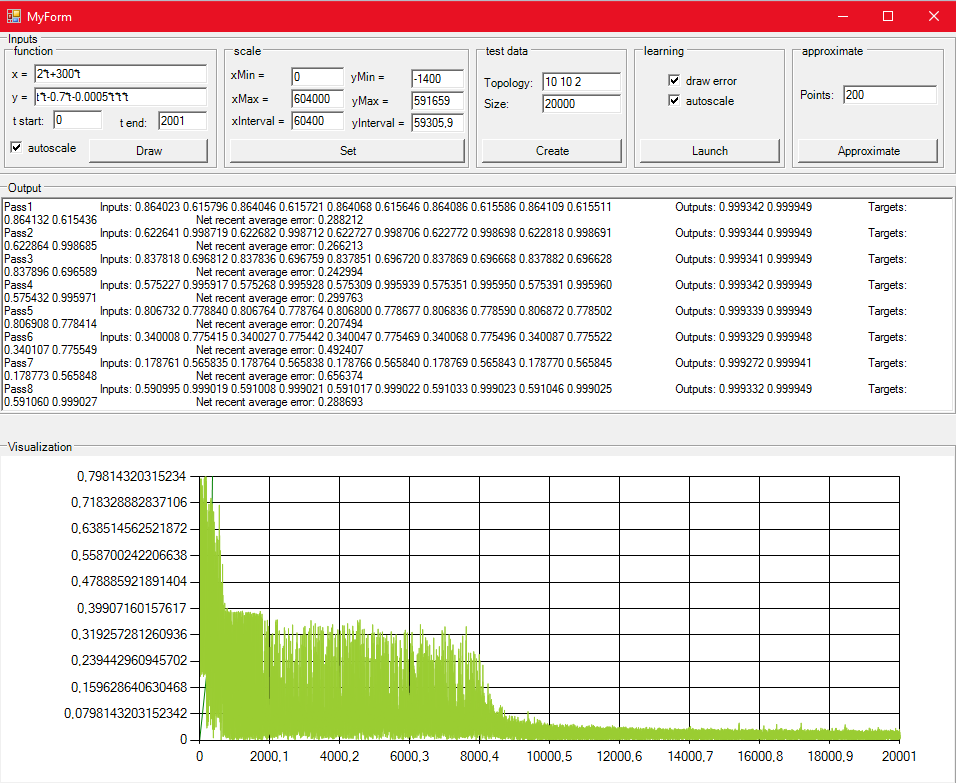
Wykres błędu dla pliku:   
trainingYCoordinate.txt – good   
Topologia: 10 10 2

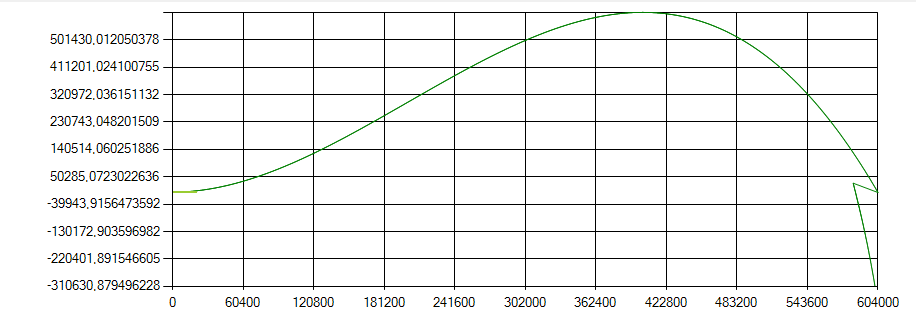


Test pokazał, że zastosowanie postaci parametrycznej funkcji pozwoli nam na użycie obu współrzędnych. Jednocześnie zauważyłem, że zwiększenie ilości neuronów na warstwie ukrytej danej efekt podobny jak dodanie drugiej warstwy. W tym przypadku sieć jednowarstwowa wystarczyła do uzyskania zadowalającego rezultatu.

1. Ilość danych testowych

Dla funkcji:  
x = 2\*t+300\*t  
y = t\*t-0.7\*t-0.0005\*t\*t\*t\*t



  
Przy takiej ilości danych uczących sieć nie uczy się wystarczająco. O ile dla danych testowych będących z przedziału danych uczących sieć działa dobrze, o tyle gdy próbujemy aproksymować funkcję, tj. pobierać punkty spoza przedziału sieć nie zwraca odpowiednich wartości. Dopiero zwiększenie ilości danych testowych do 200000 poprawia wyniki zadowalająco.  
Uwaga: po takiej zmianie nauka sieci trwa długo, przez co obliczenia wykonują się zdecydowanie dłużej.

Przykładowe screeny z działania programu:

